Complexity Measures

Worst case complexity: The number of operations the algorithm performs on the worst input.

Average Complexity: Average over the inputs x, on the number of operations the algorithm does on x.

Randomized complexity (of a randomized algorithm) on input x (denote R(x)(n)) Average over the random choices of the algorithm, of the running time of the algorithm on input x with those random choices.

Randomized complexity (of a randomized algorithm) (denote R(n)): the maximum over the inputs x of R(x)(n).

Week 4 – Some Helpful Material

Randomized Quick Sort & Lower bound & General remarks…

Continuation of Quick Sort analysis

- We are trying to get a bound on the Randomized complexity analysis, by solving the following recurrence:

$$T(n) \le \frac{2}{n}\sum_{k=1}^{n-1} T(k) + cn$$

After some algebraic tricks, we got to the following equation :

$$T(n+1) \le \frac{n+2}{n+1}T(n) + \frac{c(n+1)^2 - cn^2}{n+1} \le \frac{n+2}{n+1}T(n) + 2c$$

Randomized Complexity: Remark

Note that in randomized complexity, we say that in the worst input, we expect the running time to be R(n). But we do not say that it will be exactly that! Just on average over the random choices.

If we want to be exact, we need to say also that the probability that the running time will be much larger than R(n), is very small. (in other words, that most of the times the actual running time is concentrated around R(n), and does not exceed it by a lot.).

This is true but to show it we need too much probability theory… you will learn the probability theory required for this only next year.

Continuation of Quick Sort analysis

- We assume T(1)=O(1), and so the second term is of order n,
- And so it remains to upper bound the first term. We observe that the Harmonic series appears there:

$$H(m) = \sum_{j=1}^{m} \frac{1}{j}$$

We now want to show that

Claim: for all n $H(m) = \Theta(\log(n))$

Continuation of Quick Sort analysis

- We now open this by iteration:

$$T(n+1) \le \frac{n+2}{n+1}T(n) + 2c \le$$

$$\frac{n+2}{n+1}\left(\frac{n+1}{n}T(n-1) + 2c\right) + 2c \le$$

$$\frac{n+2}{n+1}\left(\frac{n+1}{n}\left(\frac{n}{n-1}T(n-2) + 2c\right) + 2c\right) + 2c \le$$

.....

$$\left(\frac{n+2}{n+2} + \frac{n+2}{n+1} + \frac{n+2}{n} + \frac{n+2}{n-1} + ... + \frac{n+2}{3}\right)2c + \frac{n+2}{2}T(1) =$$

$$(n+2)2c\sum_{j=3}^{n+1}\frac{1}{j} + \frac{n+2}{2}T(1) =$$

The Harmonic Series (Continued)

- We need to relate k to n. Since the number of elements in the k'th box is
- $$2^{k-1}$$
- We have

$$1+2+4+\ldots+2^{k-1}=n$$
$$2^k-1=n$$
$$k=\log(n+1)$$

And so we have

$$\log(n+1)/2 \le H(n) \le \log(n+1)$$

I leave it for you to check that this implies the claim for n by our assumption, and then for general n.

The Harmonic Series: proof of claim

- Divide the sum to boxes of size 1,2,4,8, etc. Where we assume for the moment that n is a power of 2, minus 1.

$$H(m)=\underbrace{\frac{1}{1}}+\underbrace{\frac{1}{2}+\frac{1}{3}}+\underbrace{\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}}+\ldots$$

The sum of elements in each box is at most 1 (because if the elements in the Box were all equal to the firs the sum would be one, but the elements are monotonically decreasing)

Also, the sum of the elements in each box is least ½ (because if the elements In the box were all equal to the first element after the box (e.g., ¼ after the second box) then the sum would be ½. But the elements are all bigger than that first element…)

So, we have $k/2 \le H(m) \le k$ where k is the number of boxes.

Lower bound

To complete the randomized complexity analysis…

$$T(n+1) \le (n+2)2c\sum_{j=3}^{n+1}\frac{1}{j}+\frac{n+2}{2}T(1)=$$
$$=(n+2)2c[H(n+1)-3/2]+\frac{n+2}{2}T(1)$$
$$=O(n\log n)+\Theta(n)=O(n\log n)$$

A similar analysis would show

$$T(n)\ge\frac{2}{n}\sum_{k=1}^{n-1}T(k)+c'n=\Omega(n\log(n))$$

Where instead of the 2c terms we would put c. Hence:

$$T(n)=\Theta(n\log(n))$$

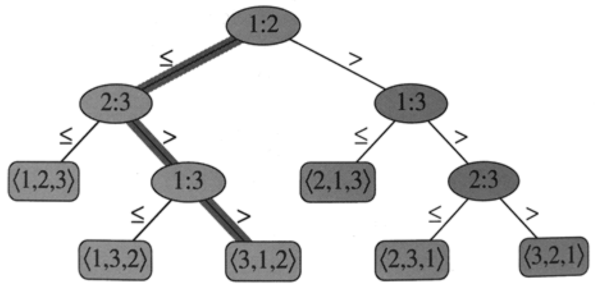
Comparison sorting – lower bound

- We want to prove a lower bound (Ω) on the worst-case complexity sorting for **ANY** sorting algorithm that uses comparisons.
- We will use the *decision tree model* to evaluate the number of comparisons that are needed in the worst case.
- Every algorithm *A* has its own decision tree T, depending on how it does the comparisons between elements.
- The length of the longest path from the root to the leaves in this tree T will determine the maximum number of comparisons that the algorithm must perform.

Sorting with comparisons

- The basic operation of all the sorting algorithms we have seen so far is the comparison between two elements: $a_i \le a_j$
- The sorted order they determine is based only on comparisons between the input elements!
- We would like to prove that any comparison sorting algorithm must make $\Omega(n \lg n)$ comparisons in the worst case to sort *n* elements (lower bound).
- Sorting without comparisons takes $\Omega(n)$ in the worst case, but we make assumptions about the input. Our lower bound will not hold in such cases.

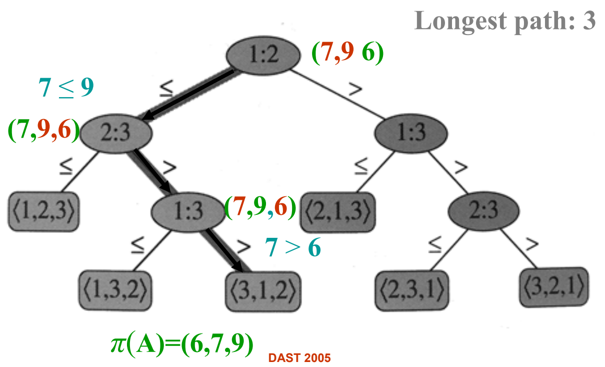
Example of Decision tree for 3 elements



Decision trees

- A decision tree is a full binary tree (a binary tree with all nodes having either two children or none) that represents the comparisons between elements that are performed by a particular algorithm.
- The tree has *internal nodes*, *leaves, and branches*:
  - Internal node: marked by two indices (*i:j*) for $1 \le i, j \le n$ *(the comparison done at the node)*
  - Branches: result of a comparison
  - $a_i \le a_j$ (left) or $a_i > a_j$ (right)
  - Leaf*:* a permutation of the input $\pi(1), \dots \pi(n)$
  - (the outcome of the sorting algorithm)

Decision tree for 3 elements



Paths in decision trees

- *The execution of sorting algorithm A on input I corresponds to tracing a path in T from the root to a leaf*
- Each internal node is associated with a *yes/no* question, regarding the input, and the two edges that are coming out of it are associated with one of the two possible answers to the question.
- The leaves are associated with one possible outcome of the tree, and no edge is coming out of them.
- At the leaf, the permutation π is the one that sorts the elements!

Comparison-based sorting algorithms

- Why “at least”? Because there might be more than one leaf with the same answer, corresponding to different ways the algorithm treats different inputs.

Decision tree computation

- The computation for an input starts at the root, and progresses down the tree from one node to the next according to the answers to the questions at the nodes.
- The computation ends when we get to a leaf.
- ANY correct algorithm MUST be able to produce each permutation of the input.
- There are *n*! permutations and they must all appear in the leafs of the tree.
- So the number of leafs in a decision tree of any correct sorting algorithm is at least n!.

Length of the longest path

- Consider a decision tree of some sorting algorithm. Let *d* be the height of the tree – the length of the longest path.
- We know: the number of leafs in a binary tree of height d is at most 2^{d}.
- On the other hand, we know that the number of leafs is at least n!.
- So we have: n! $\le$ number of leafs $\le 2^{d}$. *So* $n! \le 2^{d}$
-

Worst case complexity

- The length of the longest root-to-leaf path in the decision tree is the number of comparisons for the worst input.
- We will give a lower bound on the length of the longest path in a decision tree of any sorting algorithm.

Worst case complexity

- The worst-case number of comparisons is the length of the longest root-to-leaf path in the decision tree.
- We gave a lower bound on the length of the longest path in a decision tree of a sorting algorithm. This implies a lower bound on the worst- case number of comparisons the algorithm requires. Hence, it gives a lower bound on the worst case total number of operations of such an algorithm.
- Hence we have a lower on the worst case complexity of comparison based sorting algorithms.

Lower bound on longest path

- Therefore, $\log(n!) \le \log(2^{d}) = d$
- *But*
- *$\log(n!)=\log(n)+\log(n-1)+\log(n-2)+\ldots\log(2)+\log(1)$.*
- *Half of the terms in this sum, (n/2 terms), are larger than log(n/2).*
- *Hence we have:* $(n/2)\log(n/2) \le \log(n!)$
-
- This proves that d is Omega of nlog(n).
- This puts a lower bound on the number of comparisons (and obviously, on the total number of operations) in the worst case of a sorting algorithm.
-